VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe ve firmě

Individual Professional Practise in the Company

2009 Kryštof Laryš

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

7. 5. 2009 Kryštof Laryš

Abstrakt: Bakalářská práce se zabývá úkoly zpracovávanými v průběhu odborné praxe ve firmě Kvados, a.s., se sídlem Novoveská 1139/22, 709 00 Ostrava - Mariánské Hory. Nejprve uvádím základní popis technologií, s nimiž jsem se v průběhu praxe setkal. U nich také uvádím, jaké konkrétní úkoly jsem měl zadány k vypracování. V další části popisuji, jaké jsem volil řešení, co jsem k tomu použil za nástroje a s jakými nástrahami jsem se setkal při programování. K veškeré práci jsem používal Microsoft Visual Studio 2008 se Service Packem 1, Microsoft SQL Server 2008 a programoval jsem v jazyce C# na .NET platformě. V poslední části práce vysvětluji, jaké jsem měl znalosti před započetím praxe, jak jsem se během ní vypracoval a jaké znalosti konkrétních technologií jsem nabyl. Nakonec shrnuji, jak pro mě byla praxe přínosná a také hodnotím její celkový průběh.

Abstract: The Bachelor thesis describes various tasks the author deals with on Individual Professional Practise (hence as IPP) in Kvados a.s. Company based Novoveská 1139/22, 709 00 Ostrava - Mariánské Hory. At the beginning describes the author some basic technologies used during his IPP. Those technologies are amended by tasks needed to be solved. The solutions, used technologies and tools and suddenly arosed problems with programming comes hereinafter. During IPP Microsoft Visual Studio 2008 with Service Pack 1, Microsoft SQL Server 2008 were used. The programming was done in C# language in the .NET platform. The last part of this work illustrates the professional growth of knowledge of the author acquired during the IPP and contains a short summary of the whole IPP experience.

Klíčová slova: Windows Communication Foundation, Entity Framework, Sync Framework, MS SQL, .NET, C#, Microsoft, Objektově Relační Mapování, Servisně Orientovaná Architektura, Lucene.NET

Key-words: Windows Communication Foundation, Entity Framework, Sync Framework, MS SQL, .NET, C#, Microsoft, Object Relational Mapping, Service-Oriented Architecture, Lucene.NET

Poděkování:
Děkuji svému vedoucímu, Michalu Blaževičovi, za osobní přístup, zadané úkoly, cenné rady, pomoc a čas, který mi věnoval. Za osobní a vlídný přístup rovněž děkuji personální manažerce a konzultantce Ing. Gabriele Heliové a celému kolektivu ve firmě.

Seznam použitých zkratek a symbolů

ADO.NET	- ActiveX Data Objects .NET
ASF	- Apache Software Foundation
CA	- Certifikační Autorita
EF	- Entity Framework
GB	- Giga Byte
HTTP	- HyperText Transfer Protocol
IIS	- Internetová Informační Služba
LINQ	- Language Integrated Query
MS	- Microsoft
MSDN	- Microsoft Developer Network
MSF	- Microsoft Sync Framework
MSP	- Microsoft Student Partner
ORM	- Objektově Relační Mapper
RAM	- Random Access Memory
SOA	- Servisně Orientovaná Architektura
SP	- Service Pack
SŘBD	- Systém Řízení Báze Dat
WCF	- Windows Communication Foundation
WPF	- Windows Presentation Foundation

Obsah

1 Úvod

Cílem celé práce je nastínit průběh stáže ve firmě, kterou jsem během dvou semestrů absolvoval. Popis hlavních technologií, se kterými jsem se setkal, je rozdělen na čtyři větší části. V každé části vysvětluji základy jedné technologie nutné k alespoň základnímu pochopení principu, jelikož i samostudium bylo významnou částí praxe. Podrobněji se pak věnuji řešení konkrétních úkolů, které jsem měl zadány.

Před započetím vývoje a zpracováním přidělených úkolů bylo nejprve nutné nastudovat technologii, ve které budu vyvíjet. Jelikož znalosti programování nabyté ze školních lavic se nedají ani srovnat se znalostmi nabytými přímo z praxe, musel jsem nejprve nastudovat vše, co se dalo. V první části popisuji základy Windows Communication Foundation, kterému jsem se věnoval po celou dobu praxe. Jako druhá a základní technologie je Entity Framework, díky kterému mohu pracovat s databází a využívat přitom Windows Communication Foundation službu. Ve třetí části vysvětluji, jak pomocí Lucene.NET vytvořit extra rychlý fulltextový index nad databází a jak vyhledávat záznamy pomocí fulltextu. V poslední základní části ukazuji, že pro synchronizaci webové služby s klientskou aplikací se dá použít Sync Framework a také vysvětluji, jak na to.

Druhá velká část práce je věnována samotnému řešení úkolů a problémům, se kterými jsem se během programování setkal. Ukazuji řešení, která jsem použil, vysvětluji, z jakých zdrojů jsem čerpal a proč. Dále se snažím ukázat, proč některé úkoly neměly úspěšný výsledek a také prakticky pomocí krátkých zdrojových kódů nastiňuji některá řešení.

Nakonec porovnávám, kolik znalostí programování jsem měl před započetím praxe a kolik po dokončení. Snažím se vyjádřit míru toho, kolik jsem se během praxe naučil nových technologií a jak jsem svých nově nabytých znalostí využil v dalším životě. To nejen při výuce architektury technologie .NET či Entity Frameworku, ale i při psaní seriálu o technologii Windows Communication Foundation a další. Závěr vypovídá o tom, jak moc byla pro mě praxe prospěšná a jak to ovlivnilo můj další profesionální vývoj v oblasti programování.

2 Profil společnosti, pracovní zařazení

2.1 Historie

Historie společnosti KVADOS, a.s. sahá do roku 1992. Od svého založení se společnost vypracovala z pozice malé regionální softwarové společnosti k významnému systémovému integrátorovi s celorepublikovým působením a producentovi softwarových řešení s budovanou prodejní sítí v celé Evropě.[7]

2.2 Vývoj

V počátečních fázích rozvoje se společnost začala orientovat na distribuci produktů třetích stran. Již záhy odmítá tuto distribuci a orientuje se na vývoj a implementaci vlastních řešení. Hlavní orientací posléze zůstal vývoj komplexních informačních systémů pro obchodní a obchodně výrobní společnosti.[7]

Klíčovým produktem společnosti je od roku 1998 produkt VENTUS. Jedná se o komplexní ERP řešení, které dnes pokrývá všechny běžné finanční, logistické i distribuční procesy. Obsahuje specializovaná řešení pro celní agendy, manažerské informační systémy, CRM a řešení pro obchodní zástupce v terénu.[7]

2.3 Současnost

Společnost dále produkuje mobilní informační systém myAVIS™. Od uvedení tohoto produktu na trh patří společnost k leaderům trhu nejen v České republice, ale i Evropě. Řešení myAVIS™ pokrývá všechny běžné procesy práce mobilních pracovníků v terénu a řeší oboustrannou replikaci dat s centrálním systémem, jejich vyhodnocení a následnou integraci do provozovaných řešení typu CRM a ERP.[7]

2.4 Pracovní zařazení

Během vykonávání odborné praxe jsem byl zařazen k Michalu Blaževičovi, který je expertem na vývoj nejnovějších technologií a je považován za jednu z největších firemních kapacit v této oblasti, a to nejen mnou, ale i svými kolegy.

3 Zadané úkoly

Nepracoval jsem na žádných konkrétních interních projektech. Mně zadávané úkoly sloužily ke zjištění možností použité technologie pro účely firmy s tím, abych vždy následně vysvětlil a obhájil mnou navržené řešení a možnosti použití pro firmu.

Mé úkoly byly tedy zadávány spíše s obecným nežli konkrétním charakterem. Ke konkrétním úkolům jsem se dostával až v pozdější fázi praxe. Zpočátku jsem měl jeden z hlavních úkolů seznámit se s technologií Windows Communication Foundation (WCF), o které jsem do té doby prakticky nic netušil. V průběhu celé praxe jsem pracoval převážně s touto technologií. Další velmi rozsáhlou technologií, se kterou jsem se měl naučit pracovat, je Entity Framework, a to i ve spojení s Lucene.NET. Poslední velká část byla věnována Microsoft Sync Frameworku.

3.1 Windows Communication Foundation

3.1.1 Základní pojmy[10]

Služba je systém, který poskytuje jeden nebo více endpointů (koncových bodů), které slouží pro příjem a odesílání SOAP zpráv (messages). Služba publikuje metadata. Tvoří ji tři základní části:

- třída služby, což je její samotná implementace
- hostovací prostředí neboli místo, kde služba poběží
- jeden nebo více endpointů

Endpoint je ve službě místo, které slouží k přijímání a odesílání zpráv. Tvoří jej tři části. Adress, Binding a Contract, což se dá jednoduše zapamatovat jako ABC.

- **adress** – adresa nám říká, kde služba běží, tedy kam budou zasílány zprávy
- **binding** – říká nám, jakým způsobem bude služba komunikovat, tedy jaký komunikační protokol je zvolen, jaké kódování, ale také výběr bezpečnosti, sessions, transakcí atd.
- **contract** – specifikuje rozhraní, které služba poskytuje, metody a další. Contract je nezávislý na volbě adresy a bindingu. V tomto bodě opět vidíme, v čem spočívá síla WCF.

SOAP zprávy (messages) jsou zprávy zasílané mezi klientem a serverem (službou) a jsou nezávislé na přenosovém protokolu. Je to vlastně požadavek a odpověď po nějakém komunikačním kanále. Přesnější model posílání zpráv (messaging patterns), které WCF podporuje, je následující:

- **one way** – klient odešle zprávu službě a neočekává odpověď
- **request-response** – klient pošle požadavek a čeká na odpověď
- **duplex** – obousměrná komunikace mezi klientem i službou probíhající asynchronně (služba může vynutit spuštění metody na straně klienta)

Metadata slouží k popisu služby. Tento popis specifikuje všechny údaje důležité k tomu, aby na jejich základě mohl být nakonfigurován klient. Díky tomu klient ví, na jakém protokolu služba běží, na jaké adrese atd.

Hostovací prostředí je místo, kde služba poběží, kde bude hostována. WCF služba může být hostována například v IIS (Internetové Informační Službě), klasickém Windows procesu anebo také jako tzv. self-hosting, což může být prakticky jakákoliv aplikace (konzolová, WinForm, WPF).

Kanál (channel) je prostředí, ve kterém se přenášejí zprávy. Toto prostředí je vytvořeno v okamžiku, kdy chce klient zaslat nějakou zprávu endpointu služby.

Proxy je prostředník mezi klientem a serverem a používá se na straně klienta vždy, i když jde jen o komunikaci na jednom PC v rámci paměti. Pokud chce klient komunikovat se serverem, musí si vždy vytvořit instanci proxy třídy. Metody, které služba vypublikovala jako proveditelné, jsou pak součástí proxy třídy a klient je tak může volat.

3.1.2 Konkrétní úkoly

3.1.2.1 Základy

Mým prvním a primárním úkolem bylo nastudovat co nejvíce technologii WCF, a to hlavně základy vytvoření služby, hostování služby, práce s klientem a další. V několika dnech jsem studoval úplné základy a snažil se pochopit, o co vůbec jde. Naimplementoval jsem několik služeb hostovaných v konzolové aplikaci, WPF aplikaci, WinForms aplikaci, IIS a Windows Service. Klientské aplikace jsem převážně programoval jako konzolové aplikace nebo WinForms.

3.1.2.2 Bezpečnost

Později jsem se dostal k bezpečnosti, jak přesně zabezpečit protokol HTTP, pokud mám službu hostovanou v IIS, jak zabezpečit ostatní protokoly různými způsoby, převážně net.tcp a net.pipes. Práce s certifikáty na serveru, vytvoření testovacích certifikátů fiktivní CA, klientské certifikáty X509.

3.1.2.3 Sessions, výjimky, instancing, concurrency, throttling

Kratší část samostudia jsem věnoval sessions a pokročilému zpracování výjimek. Instancingu neboli nastavení způsobu vytváření instancí tříd služby, concurrency neboli nastavení vztahu vláken a instancí a throttlingu neboli nastavení limitu současně existujících instancí, sessions a volání.

3.1.2.4 Routery

Několik dalších dní jsem implementoval různé druhy WCF routerů, například HeaderValidationAtRouter a PassThroughRouterDuplex, které pracují na způsobu routování zpráv zasílaných z klienta na službu a naopak. Jsou to jakoby mezislužby stojící na cestě mezi klientem a službou. Některé routery stojí v cestě mezi klientem a několika službami přímo, kdy klient zašle zprávu endpointu routeru, a ten následně přepošle upravenou zprávu z klienta na službu, kterou sám vybral. Takovéto vybírání se odvíjí od konkrétně naimplementovaného scénáře. Klasický způsob užití je, když požadujeme, aby zpráva došla na aktuálně nejméně vytíženou službu, což pomocí vnitřních mechanismů musí zpracovat právě onen router. Můžeme ale použít i router, který odposlouchává kanál, kde komunikuje klient se službou.

Klient netuší, že někde na cestě je router a k tomu ani neví, že existuje více služeb a endpointů, protože svou zprávu posílá vždy na jeden cílový endpoint. Odeslaná zpráva je routerem monitorována a v případě nutnosti přeposlána na jinou službu.

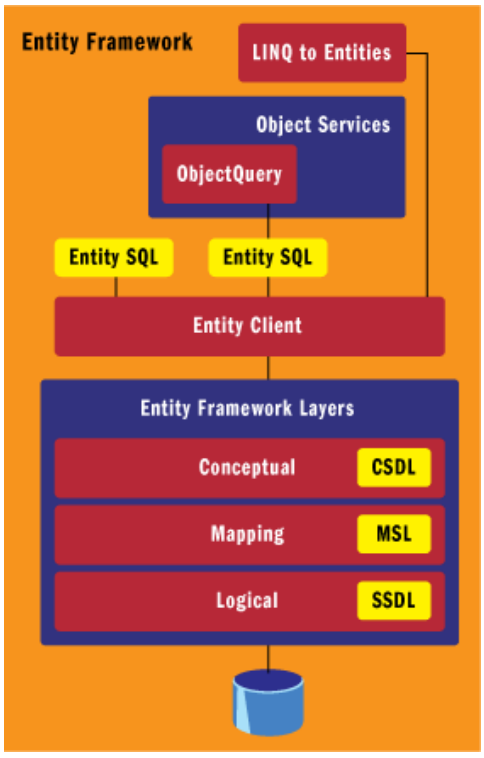
3.1.2.5 Performance Countery

Performance countery jsou systémová čidla, která nám slouží k měření spousty údajů týkajících se služby, endpointů či kontraktů. Já měl za úkol zjistit, jak se s nimi pracuje, jak se aktivují, v čem se monitorují a jak se dají přizpůsobit. Dále jsem měl zjistit, jak přesně a co se volá v systémové assembly System.ServiceModel při používání performance counterů. Nakonec jsem zjišťoval, jakou režii má moje WCF služba se zapnutými performance countery oproti vypnutým. Naimplementoval jsem si i vlastní performance counter pro měření aktuální vytíženosti procesoru.

3.2 Entity Framework

3.2.1 Popis technologie[2]

ADO.NET Entity Framework je plnohodnotný ORM z dílny firmy Microsoft. Přišel s .NET Framework 3.5 SP1 a Visual Studiem 2008 SP1. Umožňuje mapování jak 1:1, tak M:N na Entity Datový Model (EDM), což je XML soubor, který si může zkušenější programátor upravovat ručně. To je jeho podstatná výhoda oproti například LINQ to SQL, což není plnohodnotný ORM. Další velmi podstatnou výhodou je možnost použití EDM s různými databázemi, nejen MS SQL jako tomu je u LINQ to SQL. Po namapování do EDM pracujeme s tabulkami, které jsou reprezentovány třídami, řádky instancemi tříd, sloupce jako vlastnosti tříd a relace jsou asociace mezi třídami.



Obr. 3-2: Základní vrstvy Entity Frameworku[1]

Vzhledem k obecnému modelu nezávislého na SŘBD vzniká i jedna nevýhoda. EF totiž nepodporuje implicitně tzv. lazy-loading, což znamená, že se nám programátorům mírně znesnadňuje práce. Nová verze EF by ale už měla lazy-loading podporovat. Jde o metodu přednačítání dat, kdy například procházím nějakou kolekci, pak předpokládám, že při prvním průchodu se mi načte předem celá tato kolekce. EF ovšem takto nezíská referencované entity, takže je potřeba na to myslet v kódu. Pro explicit loading zavoláme metodu Load na objektu reference.

```
if (!product.CategoryReference.IsLoaded)
{
          product.CategoryReference.Load();
}
```

Pro eager loading v dotazu uvedeme, které referencované entity se mají nahrát pomocí metody Include.

```
var query = from prod in
          context.Product.Include("Category") select prod;
```

K entitám přistupujeme přes typový ObjectContext, kde můžeme použít jak LINQ to Entities či Entity SQL syntaxi,

```
using (SampleDBEntities context = new SampleDBEntities())
{
     var query = context.CreateQuery<Person>("SELECT VALUE
p from Person as p");
     var result = query.Execute(MergeOption.AppendOnly);
     foreach (Person per in result)
     {
           Console.WriteLine("{0} - {1}", per.Name,
per.GetType().Name);
     }
}
```

nebo přes Entity Clienta, kde podobně jako v klasickém ADO.NET vytvoříme EntityCommand a data čteme pomocí EntityDataReaderu.

```
EntityConnection connection = new
EntityConnection("name=EntityFrameworkDBEntities");
string eSql = "SELECT VALUE p FROM
EntityFrameworkDBEntities.Product as p";
connection.Open();
EntityCommand command = connection.CreateCommand();
command.CommandText = eSql;
EntityDataReader reader =
command.ExecuteReader(CommandBehavior.SequentialAccess);
while (reader.Read())
{
          Console.WriteLine(reader["name"]);
}
connection.Close();
```

Jak jsem v průběhu programování sám zjistil, není vůbec na škodu vyzkoušet si všechny tyto metody pro práci s daty. Entity Provider například zajišťuje velmi rychlý přístup k datům pro čtení. Pokud naopak znám syntaxi LINQu, využiji LINQ to Entities, což mi hodně zjednoduší práci nejen při psaní dotazů, ale i při práci s ObjectContextem.

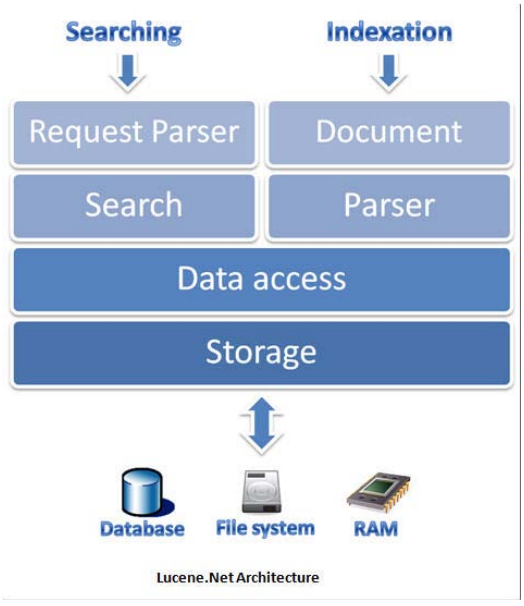
3.2.2 Konkrétní úkoly

Mým úkolem v této oblasti bylo převážně samostudium, seznámení se s technologií EF a prací s databází. Ze začátku jsem měl naimplementovat pouze základní metody pro SELECT, INSERT, UPDATE a DELETE, které znám z ADO.NET. Pracovně jsem studoval základy zhruba jeden den, mimo praxi jsem se jim však věnoval ještě spoustu dalších dnů a týdnů, věnuji se jim dodnes. Ostatně toto platí i u WCF, o kterém aktuálně publikuji podrobný seriál. Pochopitelně se tyto dvě základní technologie celou praxi prolínaly, takže když jsem vyvíjel nějakou aplikaci postavenou na WCF, většinou pracovala s databází, a to právě pomocí EF.

3.3 Lucene.NET

3.3.1 Popis technologie

Lucene.NET je portace původního Lucene z Javy do jazyka C#. Jde o velmi výkonnou, robustní a kvalitní open source knihovnu vyvíjenou pod ASF, která umožňuje fulltextové vyhledávání, a to nejen nad soubory či internetovými stránkami, ale i nad databází. Základní ideou této technologie je poskytnout opravdu velmi rychlou možnost vytvořit tzv. indexový soubor, který si tato knihovna vytvoří, a pomocí kterého pak bude vyhledávat záznamy.[6]



Obr. 3-3: Architektura Lucene.NET[9]

Jako ve většině vyhledávacích strojů i v Lucene se indexují dokumenty a jejich části (text, obrázky, objekty) prostřednictvím klíčových slov. Tato klíčová slova se pak ukládají do invertovaných (indexových) souborů. To, v čem se jednotlivé vyhledávající stroje liší, je uspořádání klíčových slov v invertovaném souboru a architektura těchto souborů. Jde o to, aby

byla zachována taková konstelace, která by umožňovala rychlé indexování a ještě rychlejší vyhledávání v bázi dokumentů. Většina vyhledávacích strojů řadí data v indexovém souboru do tvaru binárních stromů. Lucene v tomto ohledu zaujímá zcela jiný postoj. Místo toho, aby vytvářel jeden index, vytváří při indexování mnoho indexových segmentů, které pravidelně slučuje. Pro každý nový dokument Lucene vytváří nový indexový segment, hned ovšem dochází k reorganizaci segmentů spočívající ve slučování malých segmentů do velkých. Takto zůstává na relativně malém počtu indexových segmentů, takže je zachována rychlost vyhledávání. Pro indexy, které se jen málokdy použijí, je Lucene schopno vložit tyto do jednoho segmentu. Aby se zamezilo kolizím mezi čtecími a zapisovacími proudy, Lucene vždy vytváří nové soubory segmentu a maže ty staré. Podporuje inkrementální sestavování indexu, stop slova a multiuživatelský přístup. Tedy pokud se zároveň indexuje, uživatel může vyhledávat i v ještě nekompletním indexovém souboru.[8]

3.3.2 Konkrétní úkoly

Úkolům týkajících se Lucene jsem věnoval čistého času asi 14 dní. Zpočátku jsem se s technologií jen seznamoval a zjišťoval, jak pracuje Lucene v dodaném ukázkovém příkladu, kde vyhledává slova nad soubory a internetovými stránkami. Následně jsem nastudoval indexování nad databází MS SQL v podání klasického přístupu ADO.NET. Vzhledem k tomu, že je Lucene navržen programátory mluvícími a psacími výhradně anglicky, má každá kultura možnost vytvořit si vlastní analyzátor slov. Dalším úkolem tedy bylo vytvořit třídu CzechAnalyzer, která bude umět zpracovat česká slova a diakritiku. Po dokončení implementace CzechAnalyzeru jsem dostal za úkol provést indexování nad databází pomocí Entity Frameworku a nakonec převést celé toto zpracovávání na asynchronní a na straně WCF služby, doladit chyby a provést nezbytné změny v kódu.

3.4 Sync Framework

3.4.1 Popis technologie[1]

Sync Framework vydal Microsoft opět za účelem zjednodušení práce pro programátory. Tento Framework obsahuje již používané technologie jako Sync Services for ADO.NET nebo Sync Services for File Systems, ale nenutí programátora psát kód ručně, nýbrž jej generuje pomocí wizardu. Jsme tak schopni velmi jednoduše synchronizovat například soubory nebo data z databáze přes webovou službu a PDA. Vestavěný Wizard Visual Studia za nás zpacovává následující:

- Přidá reference nezbytných assembly.
- Přidá konfigurační stringy pro serverovou a klientskou část databáze.
- Přidá do solution SQL skripty a SQLUndo skripty. Vytvoří nové sloupce pro sledování změn (updatů) a vkládání (insertů), triggery k naplnění změněných sloupců, "TombStone" tabulky používané k sledování vymazaných záznamů a další. Undo skripty mohou být použity k smazání všech výše zmíněných SQL skriptů.
- Vykoná SQL skripty na SQL Serveru.
- Přidá klientskou respektive serverovou databázi.
- Vytvoří DataSet z nahrané databáze.
- Přidá "sync" control do solution v podobě lokální databázové cache. Synchronizační průvodce se dá znovu aktivovat otevřením tohoto souboru.
- Sesynchronizuje databázi pro první užití.

3.4.2 Konkrétní úkoly

Po samostudiu práce s touto technologií jsem měl vytvořit testovací databázi a zkusit ji sesynchronizovat přes webovou službu a klientskou aplikaci. Jakmile se mi to povedlo, měl jsem stejný scénář otestovat v případě, že v databázi s jednou tabulkou nebude zhruba 16 000 záznamů, ale něco přes 15 000 000.

3.4.3 Project Codename "Astoria Offline", Alpha Preview[3]

Projekt vyvíjený a založený na Sync Frameworku, ale s tím rozdílem, že při synchronizaci nepoužívá DataSety jako Sync Framework pro ukládání databáze do paměti a také místo lokálního nahrání klientské databáze vytváří Entity Datové Modely pomocí Entity Frameworku. Úkolem bylo tento projekt ozkoušet a zjistit, jak funguje.

4 Řešení úkolů

4.1 Windows Communication Foundation

4.1.1 Základy

Základy technologie WCF jsem studoval převážně z anglického MSDN, kde jsou velmi podrobně popsány i pokročilé vlastnosti, spousta zdrojových kódů a oficiálních fór. Právě nedostatek česky psané literatury o této dnes velmi využívané technologii a samotný zájem o ni mne inspiroval k tomu, abych začal publikovat vlastní seriál o WCF nazvaný „WCF pro začátečníky“, nacházející se na portálu www.NetStudent.cz. Zhruba stejně cenným zdrojem mi bylo asi pět knih v elektronickém formátu, které mi poslal na začátku praxe můj vedoucí Michal Blaževič. Chvíli po začtení se do knih, různých zdrojů a seznámení se s mechanismem WCF jsem začal s programováním své první WCF služby. Následovalo vytvoření hostovací a klientské aplikace. Zkoušel jsem využít základní tři bindingy jako wsHttpBinding, netTctBinding a netNamePipedBinding.

4.1.2 Bezpečnost[1]

Bezpečnosti jsem věnoval adekvátní úsilí, kde jsem si vyzkoušel naimplementovat různé způsoby zabezpečení. Ať už zabezpečení protokolu HTTP přes SSL, když jsem měl službu hostovanou v IIS, nebo vytváření vlastních serverových a klientských certifikátů smyšlené testovací CA. Dále také přihlašování pomocí uživatelského jména a hesla a základní zabezpečení na úrovni transportu zpráv či šifrování zpráv.

4.1.3 Sessions, výjimky, instancing, concurrency, throttling[1]

U všech dalších součástí a vlastností WCF, které se dají využít a já se je snažil naučit, jsem vesměs použil stejný postup. Vždy jsem z nějakého zdroje nastudoval příklad toho, jak se dá daná část využít a naprogramovat, nakonec jsem si naimplementoval vlastní příklad využití.

4.1.4 Routery[4]

Routery už byly celkem pokročilejší téma a byla výzva se s nimi poprat. Samotný MS vydal spoustu WCF samplů, kde jsou i předimplementovány různé druhy routerů použitelných v praxi pro variabilní scénáře. Mým cílem bylo vyzkoušet si různé druhy routerů, které jsem již popsal v zadání, ovšem nejvíce jsem se asi soustředil na Load-Balancing Router. Ten má za úkol sledovat komunikaci mezi klienty a službami a v případě vytíženosti linky nebo služby přeposlat zprávu z původního endpointu na jiný endpoint, který ale obsahuje stejný kontrakt. Zpráva musí neporušená a nezměněná dorazit i zpět na klientskou aplikaci. Klient samozřejmě netuší, co se děje na pozadí jeho právě otevřeného komunikačního kanálu a neví, že nějaký router přeposílá jeho zprávu jinam.

4.1.5 Performance Countery

U Performance counterů jsem měl za úkol nastudovat, co všechno se dá v rámci WCF měřit, v jakém programu se to dá měřit a jak se vůbec tak o v ěcountery aktivují. Brzy jsem zjistil, že nejde o nic složitého a po několika málo úpravách v konfiguračním souboru WCF služby jsem aktivoval tyto countery. Měření jsem si nechával zobrazovat v programu perfmon, který je součástí systému Windows. Přišel jsem na to, že lze měřit spoustu vlastností na úrovni

služby (ServiceModelService), jednotlivých endpointů (ServiceModelEndpoint) či dokonce každého kontraktu (ServiceModelOperation) zvlášť. Cvičně jsem si naprogramoval vlastní counter, který vypisuje aktuální zatížení procesoru v procentech, obnovované po jedné sekundě. Později bylo mým úkolem zjistit, jak countery pracují na úrovni IL kódu a co se přesně volají za metody při jejich aktivaci. Použil jsem nástroje jako ildasm k dissasemblování systémové knihovny System.ServiceModel k následné analýze a .NET Reflector pro zobrazení některých private metod, které nejsou normálně viditelné v jazyce C#. Nakonec jsem zkusil z klienta 5000x zavolat metodu služby s vypnutými a zapnutými countery, abych zjistil jejich režii. Časový rozdíl mezi zpracováním byl minimální, takže mohu jen potvrdit, že zapnuté performance countery mají velmi malou režii.

4.2 Entity Framework[1]

Část praxe, kdy jsem se věnoval technologii Entity Framework, by se dala nazvat jako druhá část. Od této chvíle jsem se začal soustředit na spojení WCF služby s databází, u kterého jsem zůstal až do konce praxe. Vytvořil jsem si několik testovacích databází a učil se, jak s nimi pracovat právě pomocí EF. Zpočátku jsem se snažil využívat Entity SQL syntaxe s ObjectContextem pro práci s entitami, později jsem už uměl pracovat s více tabulkami, loadovat cizí klíče a vytvářet komplexnější příkazy pro výpisy, vkládání, měnění a mazání. Díky znalosti EF jsem jej využil nejen v jednom školním projektu, pro osobní účely, v průběhu dalších částí praxe, ale i jako téma cvičení, která jsem jako MSP vyučoval pravidelně ve škole. V neposlední řadě jsem využil i znalost LINQ to SQL, kterou jsem pak jednoduše aplikoval vůči LINQ to Entities a hned mi došlo, jak jsou tyto technologie provázány a jak ohromně zjednodušují práci, pokud člověk ví, jak na to.

4.3 Lucene.NET

Zřejmě nejdelší čas strávený nad jednou konkrétní technologií byl právě u Lucene. Studoval jsem defaultně dodané examply a zkoušel vyhledávání nad soubory a HTML stránkami. Další příklad pro práci s databází jsem přepisoval z Visual Basicu do jazyku C#, kde se využívalo ADO.NET přístupu do databáze. Následně jsem tento příklad upravil tak, aby místo klasického ADO.NET spolupracoval s novým EF. Snad jediná nevýhoda v Lucene je taková, že programátor nemá k dispozici žádný wizard, takže si musí veškerý kód psát sám. Zároveň je ale evidentně jasné, že to je také největší výhoda, pokud kód napíši zcela správně a optimalizovaně. Svůj příklad jsem začal značně rozšiřovat podle toho, jak potřeboval můj vedoucí. Naimplementoval jsem metody pro výpis, vkládání, měnění, rušení z databáze a další. Napsal jsem základní kód, který zpracoval mou databázi a zaindexoval všechny záznamy. Dále to, co už fungovalo, jsem musel přepsat tak, aby to šlo využít ve WCF službě. Spoustu dní jsem strávil nad tím, jak udělat tzv. inkrementální rebuild už jednou vytvořeného indexu. Vše jsem musel vymyslet sám, jelikož na to nikde neexistoval žádný příklad. Když mám vytvořenou nějakou databázi a k ní indexový soubor, změním, přidám nebo smažu položku nebo několik položek z databáze a potřebuju, aby se změna projevila i do indexového souboru. Změn může být ovšem několik stovek najednou při víceuživatelském přístupu, takže není dobré dělat takový rebuild ihned při každé operaci. Vymyslel jsem tedy metodu, která se o inkrementální rebuild postarala, jenže naneštěstí byla její režie časově několikrát vyšší, než vytvoření celého indexu úplně znovu. Důvod byl jasný. Metoda musí zjistit, jestli aktuální položka, na které se nacházím v databázi, je v indexu nebo ne. Pokud ne, musím ji přidat do indexu, pokud ano, přeskočím ji. Zároveň ale musím kontrolovat, zda-li nemám některé položky pouze v indexu, ale v databázi již neexistují. V takovém případě je musím z indexu odstranit. Poslední a nejnáročnější na

implementaci i časově je část, kdy kontroluji, zda položka v databázi i indexu obsahuje stejná data, nebo se data změnila. V tom případě je musím v indexu přepsat tak, aby byla shodná s databázovými daty. Jak jsem již zmínil, tato metoda hromadného inkrementálního rebuildu nevedla k zdárnému konci, a proto bylo potřeba vymyslet schůdnější řešení. Mezitím jsem se věnoval přepisu a implementaci CzechAnalyzeru, který byl schopný korektně zaindexovat i české výrazy, diakritiku. Testoval jsem vytváření indexu nad tabulkou s přibližně 15000 záznamy, což trvalo kolem 60 sekund. Bylo mi řečeno, že je to několikrát rychlejší řešení, než má vestavěné MS SQL. Když jsem později přišel s novou verzí Lucene, která právě vyšla, mírně jsem musel upravit již mnou vytvořené metody, ovšem výsledek předčiločekávání nejen moje, ale také očekávání experta, mého vedoucího Michala Blaževiče. Celá tabulka se zaindexovala za pouhých 5 sekund. V tomto případě už „stará“ metoda na inkrementální rebuild ztratila smysl úplně. Úplně jsme tedy přehodnotili strategii a dohodli jsme se, že nejlepší možnost, jak provést inkrementální rebuild, bude jeho spouštění automaticky, a to po každé operaci. Každou metodu jsem musel ale ještě více strukturovat a upravit tak, abych mohl použít zamykání objektů a nerušily se přístupy do indexového souboru. Řešení dopadlo tak, že se při databázových operacích vždy uzamkl společný objekt, který se po dokončení opět odemkl a zároveň se v ThreadPoolu na pozadí provedl rebuild. Zjistili jsme, že je to nejlepší možný způsob, který nemá téměř žádnou režii. Když totiž vkládám záznam do databáze, vím, že jej musím vždy vložit i do indexu. Stejně tak když záznam mažu, musím jej vždy smazat i z indexového souboru. Když záznam změním, změním jej i v indexovém souboru. Zkrátka se mi místo velké metody rebuildu, která projíždí celý indexový soubor, stane velmi jednoduché a hlavně rychlé řešení tohoto problému. Dále jsem už jen vylaďoval službu do přijatelné podoby, ošetřoval výjimky a zavedl jsem automatickou časomíru vytváření indexu, která se ukládala do log souboru. Službu jsem posléze nechal hostovat ve Windows Service. Poslední větší detail, který jsem řešil, byl způsob zaslání vyhledaných dat ze služby na klienta. Problém je v objemu dat. Když přes službu posílám celý DataSet, je to sice jednodušší pro implementaci na straně klienta, ale ohromně to zvětšuje nároky na přenosovou kapacitu. Proto jsem vytvořil i druhou metodu pro vyhledávání, která vyhledaná data uložila a poslala přes generickou kolekci obsahující tabulku jako typ.

4.4 Sync Framework[5]

Jeden z mých posledních úkolů bylo samostudium Microsoft Sync Frameworku. Nejprve jsem studoval ukázkové příklady, jak vůbec synchronizace funguje, a to převážně na synchronizaci souborů. Následně jsem použil již vytvořenou databázi s přibližně 16 000 řádky v jedné tabulce a provedl jsem synchronizaci této databáze na straně webové služby s klientskou aplikací na druhé straně pomocí vestavěného Wizardu Visual Studia. Prozkoumal jsem vygenerovaný zdrojový kód a upravil jej tak, abych byl schopen provést vlastní synchronizaci v klientské WinForm aplikaci. V druhé části úkolu jsem měl vyzkoušet synchronizaci webové služby s klientskou aplikací nad databází s tabulkou o přibližně 15 000 000 záznamů. Vytvořil jsem si tedy nejprve aplikaci, která mi tak obrovskou databázi vygenerovala. V tabulce jsem založil 6 sloupců. První sloupec byl auto-increment primární klíč, dalších 5 sloupců byly náhodně vytvořené stringy o velikosti přesně 10 znaků. Pro generování záznamů jsem použil tento kód:

```
using System.Data.SqlClient;
using System;

namespace FillDBData
{
```

```
        class Program
        {
            private static readonly SqlConnection conn = new
SqlConnection(@"Data Source=.\SQLEXPRESS;Initial
Catalog=BigTest;Integrated Security=True;");
            private static readonly SqlCommand commInsert =
conn.CreateCommand();

            static void Main()
            {
                commInsert.Connection.Open();
                var random = new Random();
                for (int i = 0; i <= 15000000; i++)
                {
                    int Column1 = random.Next(1000000000,
2000000000);
                    int Column2 = random.Next(1000000000,
2000000000);
                    int Column3 = random.Next(1000000000,
2000000000);
                    int Column4 = random.Next(1000000000,
2000000000);
                    int Column5 = random.Next(1000000000,
2000000000);

                    commInsert.CommandText =
                    string.Format(
                        "INSERT INTO [Test] (Column1, Column2,
Column3, Column4, Column5) VALUES ({0}, {1}, {2}, {3}, {4})",
                        Column1, Column2, Column3, Column4,
Column5);

                    commInsert.ExecuteNonQuery();

                    if (i%1000 == 0)
                    {
                        Console.WriteLine("Actual position:
{0:N0}", i);
                    }
                }
                commInsert.Connection.Close();
            }
        }
    }
```

Vykonání kódu trvalo necelé 3 hodiny. V průměru se vygenerovalo téměř 2000 záznamů za sekundu, ke konci se ale průběh mírně zpomalil, jelikož po zaplnění paměti RAM 1.5 GB daty z databáze se začalo swapovat na disk. Stejný příklad jsem naimplementoval ještě pomocí LINQ to SQL, abych otestoval, zda bude zápis rychlejší nebo pomalejší, ale podle očekávání byl zápis pomalejší. Databáze jako celek měla na disku 2.0 GB. Když jsem po úspěšném generování databáze zkusil její synchronizaci, setkal jsem se hned se dvěma

problémy. Nejprve vypršel Timeout na straně webové služby, který jsem byl ale schopen ručně přenastavit. Problém, přes který jsem se nedostal, byla výjimka OutOfMemoryException. Wizard při synchronizaci tak velkého počtu položek naprosto selhal a nedalo se s tím nic dělat. Zásadní problém byl v tom, že synchronizace probíhá pomocí načítání databáze do odpojeného DataSetu, který je ovšem celý uložený v paměti. Pokud však tato paměť přeteče, nezačne se swapovat na disk, ale wizard selže. Proč? Podle MS prozatímní nedostatek. Zkoušel jsem to vyřešit různými způsoby, ale ani jeden nebyl úspěšný. Navíc jsem se na oficiálních MSDN fórech dočetl, že MS o tomto problému ví, a že to je nejdůležitější problém, který se firma bude snažit vyřešit v nově chystané verzi Sync Frameworku, která ale spatří světlo světa až někdy v polovině roku 2009. Předpoklad mého vedoucího, že synchronizace při tak velkém počtu položek selže, se tedy jen potvrdil. Ještě jeden konkrétní nápad, jak vyřešit OutOfMemoryException, bylo zkusit projekt Astoria Offline.

4.4.1 Project Codename "Astoria Offline", Alpha Preview[2]

Po zjištění, že tento projekt nepracuje s DataSety, ale s EDM, jsem zkusil zpracovat malou databázi. Výsledek byl pro mě docela dost překvapivý. Synchronizace asi 200 položek trvala několik sekund, což bylo pro databázi s několika miliony záznamů naprosto nepřijatelné. Před samotným použitím velké databáze jsem si pročítal podrobný walkthrough, který byl k projektu přiložen. V sekci se známými problémy a nedostatky byl jako první uveden nadpis „No batching“. V této chvíli jsem už věděl, že jde jen o prozatímní další slepou uličku a nezbývá nic jiného, než čekat na nový release Sync Frameworku.

5 Závěr

5.1 Získané znalosti

V průběhu celé praxe jsem získal obrovské množství znalostí. Ihned po prvním dni, kdy jsem jen instaloval potřebný software, jsem se začal učit technologii Windows Communication Foundation. Okamžitě jsem se zahltil velkým množstvím nových poznatků a znalostí, které jsem potřeboval pro další vývoj. Obecně platí, že téměř vše, co jsem během stáže programoval, jsem předem vůbec neznal. Předpokládala se pouze znalost jazyka C# na .NET platformě, což bylo ale pochopitelně bráno jako samozřejmost. Druhou zásadní technologií, se kterou jsem se v průběhu praxe setkal, byl Entity Framework. O tom jsem rovněž předtím nic nevěděl a není se ani čemu divit, jelikož je to nová technologie vydaná až se Service Packem 1 do Visual Studia 2008. Jediná technologie ne přímo od firmy Microsoft byla Lucene.NET vyvíjená pod Apache Software Foundation. Při jejím studiu jsem tak dostal možnost více programovat v klasickém C# kódu, samozřejmě ale v práci s Lucene. Poslední velkou technologií byl MSF (Sync Framework), který je také novinkou nedávno vyšlou. Speciální balík se musí doinstalovat do Visual Studia, a pak už se může využívat možností synchronizace. Tato kapitola pro mě nebyla nijak extrémně složitá, naopak byla velmi zajímavá a ke konci celé praxe jsem viděl, jak se všechny tyto výše uvedené technologie krásně prolínají a v praxi využívají.

5.2 Chybějící znalosti

Znalosti, se kterými jsem na praxi přišel, požadavkům praxe příliš nepostačovaly. Znal jsem relativně dobře architekturu technologie .NET, ve které jsem s ročním předstihem absolvoval stejnojmenný předmět. Jazyk C# mi rozhodně nebyl cizí, ba naopak. To bylo ale tak zhruba vše, co jsem doposud uměl. Založit a naprogramovat jednoduchou konzolovou či WinForm aplikaci. Musel jsem tedy hodně času věnovat samostudiu, kdy jsem se seznamoval se všemi technologiemi, o kterých jsem do té doby nic nevěděl. Příklady těch hlavních, které jsem se musel učit, byly Windows Communication Foundation, Entity Framework, Lucene.NET, Sync Framework a další. Rozhodně jsem si upevnil znalosti i ve spoustě základních odvětví .NETu, jako například práci s LINQ, ADO.NET a v podstatě ve všem, co jsem již uměl.

5.3 Výsledky a celkové zhodnocení

Všechny zadané úkoly jsem splnil a prokonzultoval s vedoucím Michalem Blaževičem. Pokud se na praxi podívám jako na celek, můžu po jejím úspěšném ukončení konstatovat s jistotou několik skutečností. Praxe pro mne byla více než hodně prospěšná a inspirující. Na druhou stranu musím říct, že odpracovat vedle studijních povinnostíještě 50 dní po 8 hodinách není zrovna malá dávka. To ale neznamená, že se to nedalo zvládnout, ba naopak. Člověk si však musel dobře předem naplánovat celý semestr, protože času bylo opravdu málo.

Jsem přesvědčen, že získané zkušenosti využiji také v budoucnu. Technologie, se kterými jsem na praxi pracoval, využívám již v současné době. Jak v rámci pravidelných středečních cvičení, která jsem pořádal za účelem probírání vybraných témat z architektury technologie .NET, tak při psaní seriálu o technologii Windows Communication Foundation na portálu NetStudent jako nynější Microsoft Student Partner. Z tohoto i celkového hlediska hodnotím praxi a její přínos jen a jen kladně.

Literatura

1. *MSDN Library for Visual Studio 2008* [online].
   URL: <http://msdn2.microsoft.com> [citováno 3. března 2009].
2. *Microsoft Forums* [online].
   URL: <http://forums.microsoft.com> [citováno 15. března 2009].
3. *Wikipedia, the free encyclopedia* [online].
   URL: <http://en.wikipedia.org/wiki> [citováno 11. března 2009].
4. *CodePlex - Open Source Project Hosting* [online].
   URL: <http://www.codeplex.com> [citováno 20. března 2009].
5. *CodeGuru* [online].
   URL: <http://www.codeguru.com> [citováno 20. března 2009].
6. *Lucene.Net* [online].
   URL: <http://incubator.apache.org/lucene.net> [citováno 17. dubna 2009].
7. *KVADOS, a.s.* [online].
   URL: <http://www.kvados.cz/Profil/tabid/53/Default.aspx> [citováno 1. dubna 2009].
8. NOVOTNÝ, Tomáš. *lucene* [online].
   URL: <http://nb.vse.cz/~zelenyj/it442/eseje/xnovt16/final.htm> [citováno 17. dubna 2009].
9. *Blogs - ASP.NET Weblogs* [online].
   URL: <http://weblogs.asp.net> [citováno 5. dubna 2009].
10. LARYŠ, Kryštof. *WCF pro začátečníky – 1. díl: teorie, základní pojmy* [online].
    URL: <http://www.netstudent.cz/Články/tabid/56/articleType/ArticleView/articleId/224/WCF-pro-zatenky--1-dl-teorie-zkladn-pojmy.aspx> [citováno 2. března 2009].